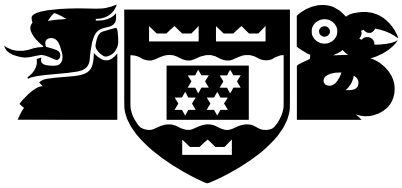
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

A Domain-Customised Business Process Model for Postal Services

Ashleigh Cains (300198169)
Alexandre Sawczuk da Silva (300193950)

October 3, 2013

Submitted in partial fulfilment of the requirements for
SWEN424 — Model-Driven Development.

Abstract

This report details a project which explored the concept of Model-driven Development (MDD). This was done by creating a visual language that enabled the modelling of business processes in a mail delivery company. To that end, a tool called Generic Modelling Environment (GME) was used, and a metamodel describing the language was created. Subsequently, the sample business process of requesting a delivery was modelled in the newly created language. Finally, an interpreter was developed and the model was translated into a Java structure using the interface provided by GME. The proof-of-concept developed in this project demonstrates the usefulness of the MDD paradigm, which enables greater involvement of non-technical users and leads to faster production of code.

1 Introduction

This report presents the worked performed and the knowledge gained during the team project for SWEN424, the Software Engineering course on Model-Driven Development offered by Victoria University of Wellington in 2013. In this project, the team developed the proof-of-concept for a visual modelling language intended to be applied when creating Business Process Models (BPMs) for the domain of a mail delivery company.

First the concepts of Model-Driven Development and Business Process Models are introduced. Then the business case used as the basis for this project is explained and the development process of the proof-of-concept is examined, with a focus on the modelling tool employed. Finally, the conclusions reached by the team about the development process are laid out.

2 Model-Driven Development

Before the project can be discussed, the concept of Model-Driven development must be explained. Model-driven development is the notion that a system can be described in terms of a model that is later transformed into the system [11]. Essentially, this allows users to build systems by modelling them in an intuitive, visual language and then transforming them into a working textual representation. The transformation step generates some or all of the system implementation in the target language/representation. When the right balance between abstraction and simplification is struck, it is possible to reduce the complexity of the problem being modeled without overlooking any of its critical aspects, a goal that has motivated computer scientists and software engineers for many decades [6].

An invaluable benefit of successful model-driven development is the automation it allows, which translates to much higher levels of productivity and more accurate end-results [12]. Specifically, this automation has to do with the generation of programs from models, as well as the verification of model correctness. These techniques are growing in maturity and as a result are being increasingly used by the software engineering community, leading to the development of industry standards.

3 Business Process Models

Another fundamental concept in this project is that of a Business Process Model. A Business Process Model, often shortened to a BPM, is a graphical way of displaying the processes utilised by a business. From the simple flow charts used at the beginning of the 20th century, BPMs later motivated the creation of a dedicated notation, the BPMN (Business Process modelling and Notation) [7].

A BPM is a valuable technique for many reasons, including the following:

1. It provides a straightforward visual representation of the processes within a business.
2. The visual notation, in its turn, enables the identification of common patterns found in business processes [9], leading to better business understanding and potentially better decision-making.
3. BPMs provide stakeholders with a communication tool that does not require technical knowledge in order to be used, yet is precise enough to convey meaningful specifications to the developers in charge of constructing the system.

The point regarding stakeholder communication is especially important. As shown by Al-Rawas and Eastbrook [4], communication problems among stakeholders are the greatest causes for failures and delays in engineering projects. More specifically, their study shows that one area of difficulty is the translation of natural language requirements into some form of representational objects. The problem is that once the requirements are translated into the new representation, many of the stakeholders are no longer able to understand the notations employed by business analysts. It is in this context that the advantages of utilising models become apparent.

A model that is understandable to non-technical stakeholders allows them to participate in the process of formalising requirements for business processes. This first-hand involvement, in its turn, increases the likelihood of creating specifications that are based on correctly interpreted requirements. Ultimately, achieving accurate specifications earlier in the analysis period of a project may lead to increased productivity, since models are likely to need less reworking.

The benefits of employing BPMs are evidenced by the increased number of companies across the world seeking to employ this technique as a core business practice [8]. Despite the necessity to configure BPMs for each particular business, which can be costly, research is being done towards less labour-intensive solutions [13]. BPMs can be applied in a wide variety of domains, including the one discussed next.

4 Selected Domain and Idea Being Modelled

The domain selected for this project was that of a postal company, more specifically the server-side of the online services provided by the company. These services are not typically large, so it is quite feasible to model them. Additionally, they operate on source code that can be generated based on a BPM model, making them ideal candidates for the Model-Driven Development approach.

The New Zealand Post was chosen as an example company for this project, since it offers a myriad of services, both online and offline. A visit to the companys website [3] reveals that the following online services are offered, among others:

- Online shopping
- Requesting a delivery online
- Setting up mail redirection
- Applying for a Post Office box
- Booking an urgent/overnight courier
- Sending money within New Zealand
- Sending money internationally
- Applying for 18+ (proof of age) cards
- Submitting United Kingdom passport applications

The modelling language created in this project was intended to be applicable to the definition of any online service offered by a postal company. However, the proof-of-concept model developed in this project was that of an online delivery request performed by a post service user. The delivery request process starts by requesting that the user specify the

pickup and drop-off locations for the item to be delivered. Then, that information is validated and the user proceeds to pay for the delivery. Once the payment transaction is complete, the system proceeds to determine which transportation mode should be employed for delivering the item at the drop-off location (it is assumed that items are picked up always using the same transportation mode). The final step is to schedule the delivery according to its transportation mode. Once the domain was selected for modelling, the next steps were to select an adequate modelling tool, as discussed in the following section.

5 Selected Modelling Tool

The modelling tool selected for this project was the Generic modelling Environment (GME), developed by the Institute for Software Integrated Systems at Vanderbilt University (TN, United States). This tool was chosen by the team because it was perceived to be a stable and well-established program with a wealth of online documentation and an intuitive user interface. The installation process is straightforward, as it consists of simply downloading the installer from the GME website [2] according to your version of Windows (32 or 64-bit), running it and following the wizard instructions.

GME provides an environment that enables the customisation of modelling languages for multiple domains [10]. This is achieved by first defining metamodels that encode the domain-specific limitations of a given modelling language, then defining the actual models choosing the metamodel with the appropriate base concepts. Any additional constraints that cannot be visually modeled are represented as Object Constraint Language (OCL) statements, as GME provides constraint-checking capabilities.

Notably, GME is capable of synthesizing output code from a model due to its architecture based on COM (Component Object Model) technology, an interface standard created by Microsoft for software components [5]. This language-agnostic interface is supported by major programming languages such as C++ and Java, enabling different types of code artifacts to be generated from a GME model according to the users needs.

This tool was chosen by the team because it was perceived to be a stable and well-established program with a wealth of online documentation and an intuitive user interface. Screenshots showing the functionality of the tool are presented in the following sections. The installation process is straightforward, as it consists of simply downloading the installer from the GME website [2], running it and following the wizard instructions.

GME is intended to be used for all stages of model-driven development, from the definition of a modelling language to the creation of a mechanism to translate models into the desired output. More specifically, GME supports the following steps:

- The creation of a *metamodel*, which establishes the features, notation and constraints of the language to be used in the creation of models for a given domain. This is accomplished through GME's metamodel perspective.
- The creation of a *model*, which is intended to represent a conceptual abstraction of a chosen domain. The modelling process can be accomplished through GME's model perspective, the first step being the selection of the modelling language to be employed (i.e. a metamodel previously created using GME).
- The creation of an *interpreter*, which is used to convert the visual model into a textual representation such as XML, SQL, or a programming language. The interpreter is meant to be created using a programming language, translating the visual models created in GME through one of the two standard interfaces provided (BON or COM –

BON is discussed in more detail later). While GME provides the capability of registering an interpreter so that it can be invoked from within it, GME itself does not provide a development environment for writing the interpreter code.

However, before proceeding to the first step (the creation of a metamodel), it was imperative to select a modelling language. The language was based on the Business Process Modelling Notation, an industry standard visual language explained in the next section.

6 The Business Process Modelling Notation

BPMN (Business Process modelling and Notation) was selected as the visual notation for the models in this project. The introductory BPM guide published by IBM [14] presents the concepts behind this notation in a very structured and detailed fashion, therefore this section is based on the information contained in that document.

In the last decade, a non-profit organisation of software industry leaders called BPMI (Business Process Management Initiative) developed a standard BPM graphical notation that incorporates elements of traditional flowcharts. The objective of this standardisation is to unify the understanding of business analysts and developers involved in this specific field of modelling. Once the common understanding exists, process owners are capable of following the evolution of the models and assist in the correction of mistakes.

The visual elements in BPMN were intentionally designed to be similar to a flowchart diagram, since that is a well-known notation to business analysts and other modelers. The four types of elements offered by this notation are *flow objects*, *connecting objects*, *swimlanes* and *artifacts*:

- **Flow objects** are the core objects in a business process, and include events (they happen during the course of a business process, and affect the process flow), activities (represent the tasks and sub-processes a company generally performs) and gateways (control the process flow, traditional path decisions, and the merging of paths). Visual representations for events, activities and gateways are shown in figures 1, 2, and 3, respectively.

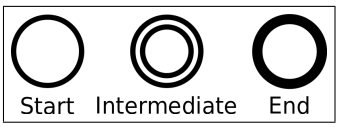


Figure 1: Representation of BPMN event types [1].

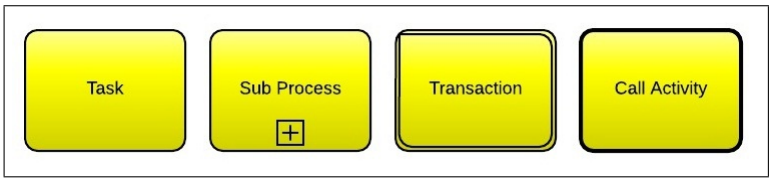


Figure 2: Representation of BPMN activity types [1].

- **Connecting objects** are responsible for actually linking the flow objects into a process. They can be sequence flows (for showing the order of activities), message flows (shows messages exchanged between separate business participants), and associations (show input/output of activities). Visual representations for connecting objects are shown in figure 4.

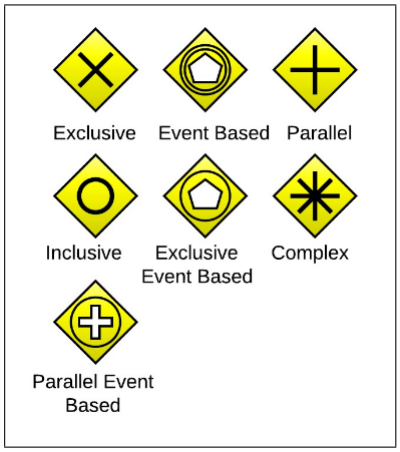


Figure 3: Representation of BPMN gateway types [1].

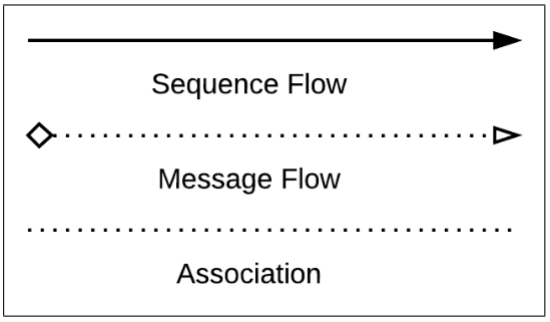


Figure 4: Representation of BPMN connecting object types [1].

- **Swimlanes** enable the organisation of activities into different groups, according to their meaning or functionality. They can be pools (basic container) or lanes (sub-partitions of pools for further organisation).
- **Artifacts** are diagram extensions that are used depending on the context of the process being modeled. They can be data objects (to model required data for activities), groups (for documentation purposes), and annotations (additional text information or comments).

Even though BPMN is compact, the combination of all its standardised elements results in a rich and expressive modelling language. The time restriction of this project did not allow for the full coverage of BPMN concepts, therefore the team concentrated on the utilisation of a subset of this notation which was smaller but still expressive enough for a proof-of-concept. The next section describes this subset in more detail.

7 Selected Notation Subset

The notation elements presented in this section are a BPMN subset chosen for this particular project. The meaning of each element type is explained in the hierarchical outline that follows. The reader might find it useful to refer to figures 1, 2, 3 and 4 when examining this section:

- **Flow Objects**
  - **Event Types**
    - **Start:** acts as a process trigger, indicated by a single narrow bordered circle.
    - **Intermediate:** represents something that happens between the start and end events; is indicated by a double bordered circle. For example, a task could flow to an event that throws a message across to another pool, where a subsequent event waits to catch the response before continuing.
    - **End:** represents the result of a process. It is indicated by a single thick or bold bordered circle.
  - **Activity Types**
    - **Task:** is a unit of work, the job to be performed.
    - **Sub-process:** used to hide or reveal additional levels of business process detail. Has its own self-contained start and end events; flow objects; connecting objects; and artifacts.
    - **Transaction:** is a set of activities that logically belong together; it might follow a specified transaction protocol (involves messaging).
  - **Gateway Types**
    - **Exclusive:** when splitting, it routes the sequence flow to exactly one of the outgoing branches. When merging, it awaits one incoming branch to complete before triggering the outgoing flow.
    - **Inclusive:** when splitting, one or more branches are activated. All active incoming branches must complete before merging.
    - **Event-based:** is always followed by catching events or receive tasks. Sequence flow is routed to the subsequent event/task which happens first.
    - **Exclusive event-based:** each occurrence of a subsequent event starts a new process instance.
- **Connecting Objects**
  - **Message Flows:** symbolize information flow across organizational boundaries and can be attached to pools, activities, or message events.
  - **Sequence Flows:** define the order of execution of activities.

Some of the elements in the generic hierarchy above were then customised by the team to meet the necessities of a model that abstracts postal company services. The following outline presents descriptions only for the customised elements:

- **Event Types**
  - **End**
    - **Error:** a specialised version of the end event was created to denote an error in the execution of the process which leads to its termination.
- **Activity Types**
  - **Task**
    - **Delivery:** represents the task of scheduling the delivery of an item to a certain address. Deliveries offer different transportation mode options: by car, by boat, by bike, and by plane.

  - **Transaction**
    - **Input:** an input retrieval transaction. It can be as simple as retrieving content from one field or as complex as retrieving large forms and uploaded documents.
    - **Payment:** a payment transaction. It debits a certain amount of money from the customer account using the credit card information provided.
    - **Money Transfer:** a money transfer transaction. Sends a certain amount of money from one bank account to another.
    - **Third-Party Transaction:** an exchange between the post company and a third-party service.

- **Gateway Types**
  - **Exclusive**
    - **Determine Transportation Mode:** determines how an item should be transported for delivery.
    - **Validate Input:** determines whether the input provided by the user was valid.
    - **Validate Payment:** determines whether a user payment was valid.
    - **Validate Money Transfer:** determines whether a money transfer was valid.
    - **Validate Third-Party Transaction:** determines whether a transaction that depends on a third-party service was valid.

These notation elements were incorporated into the metamodel, as discussed next.

8 Creating the Metamodel

The metamodel was created in GME by starting a new project using the MetaGME paradigm, which is especially designed for the definition of modelling languages. It was structured following the BPM notation described in the previous section, resulting in the class diagram shown in figure 5.

The picture shows the metamodel was developed according to the BPM hierarchy described earlier. Fundamentally, a model contains connections and flow objects. Connections can either show the general flow of a process or the path of event messages, while flow objects can be the activities, gateways and events (e.g. start, end) described earlier. Once developed, the metamodel was interpreted and registered using the MetaGME interpreter (done by clicking on the cog icon). This is a necessary step for making this paradigm – the modelling language – available in GME for the subsequent creation of models.

9 Addition of Model Constraints

In addition to the metamodel, rules can be written to ensure a model's validity. These rules are expressed in a rich language called OCL (Object Constraint Language) which is defined as part of the standard UML notation. OCL is based on sentences that are either true or false and must evaluate to "true" in order to satisfy a constraint. For this project, the team has included constraints using GME, as it supports OCL 2.0. An OCL expression becomes a constraint when it is specified in the metamodel attached to the metaobject it applies to.

To add constraints in GME, the ParadigmSheet that contains the metamodel must be open. From there the Constraints aspect needs to be selected and a constraint inserted into

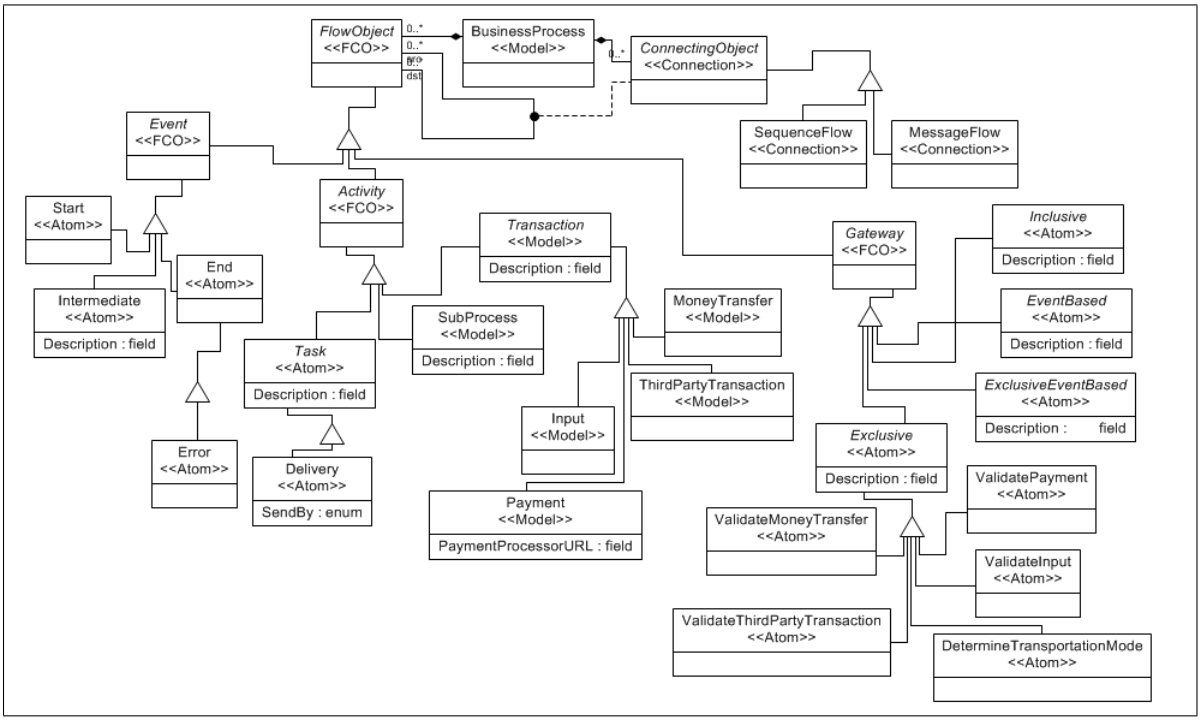


Figure 5: Screenshot BPMN metamodel.

the metamodel. An association needs to be made between the constraint and the metaobject by drawing a connection between the two so a context can be given to the OCL expression. In the attributes panel, a name, description and the expression can be added to the constraint. From there the metamodel can be reinterpreted and registered to test the constraint in the model [2].

An example of a constraint used in the project was based on the BPMN rules of:

1. A start event cannot have an incoming connecting object.
2. A start event must have an outgoing connecting object.

These rules can be translated into an OCL constraint attached to the Start metaobject in the metamodel and can be described by this expression:

self.attachingConnections("dst",Start)->size==0 &&
self.attachingConnections("src",Start)->size==1

In plain english, this expression evaluates to the set of all connections attached to Start objects and have Start as the destination of the connection, must have a size of zero. This satisfies the first of the two BPMN rules stated above. The second part of the OCL expression states that the set of all connections attached to Start objects and have Start as the source of the connection, must have a size of 1.

This process of adding constraints has been repeated for a subset of BPMN rules that apply to the selected notation subset discussed earlier. These rules have been selected to ensure the basic structure of a model based on BMPN is valid.

10 Creating the Model

As a proof-of-concept for the customised BPM language, a sample online delivery request was created. As shown in figure 6, it uses transactions for the retrieval of input and the

collection of payment, as well as delivery scheduling tasks. Decisions on which transport mode to be used when delivering an item are made using a gateway, and so it the validation of input. If there are any errors when making these decisions, the process is terminated with an Error event; otherwise, it successfully reaches the End event.

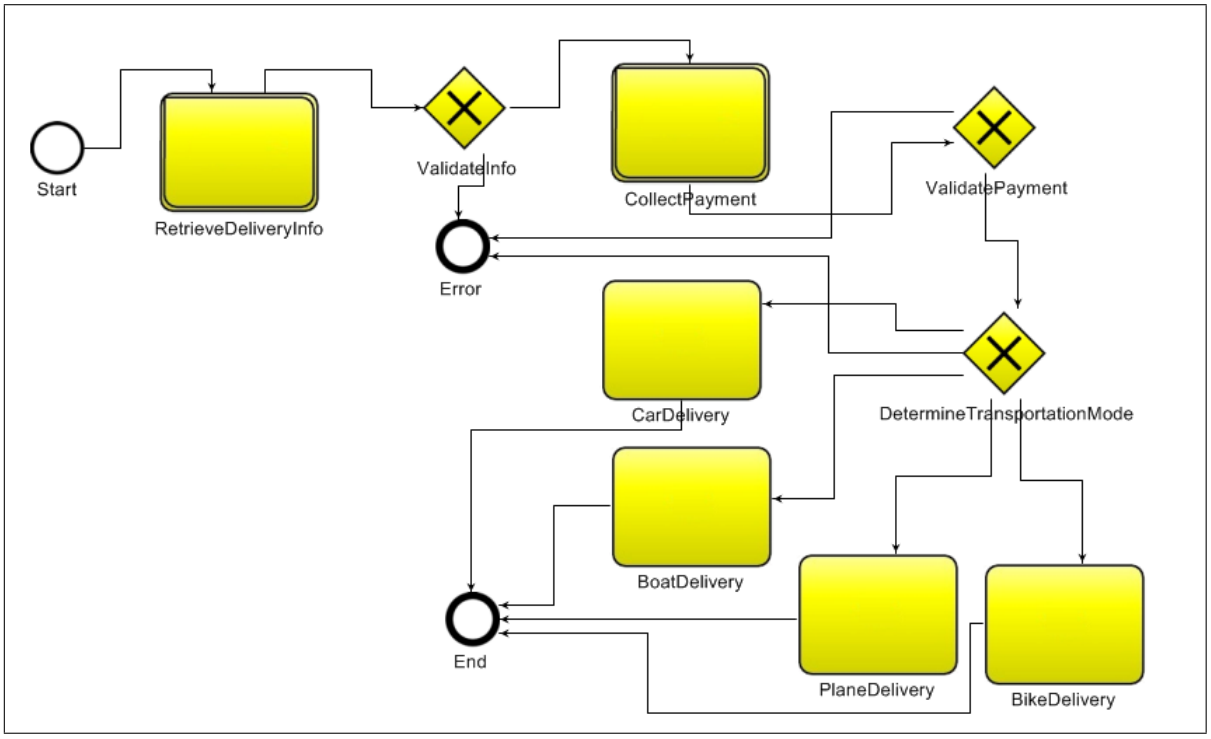


Figure 6: Screenshot of online delivery request model.

An interesting feature of GME is that it allows custom icons to be used when representing each entity in a model. This can be accomplished by providing an icon path and name to each entity through the preferences panel. The icons presented earlier were incorporated into this model, making it more easily interpretable.

11 Model Translation

Java was selected as the target language for the code implementation of the customised BPM framework. Classes and interfaces were defined to mirror the BPM hierarchy created in the metamodel, except for the connections, which were implemented as references from one object to the next. In order to translate the GME model into a correspondent Java structure, an interpreter was written.

As mentioned earlier, GME offers the Builder Object Network (BON) interface, a standardised representation of models that is supported in both C and Java. This interface enables code access to all entities in a model, as shown in figure 7, including their instance names and attribute values. In order to run an interpreter from GME and translate the model from the BON interface to the code representation, it is necessary to register an interpreter. This is accomplished by ensuring that the main class of the the Java interpreter implements the BONComponent interface and by running JavaCompRegister.exe. Once that is done, the interpreter can be run from GME by clicking on the coffee cup icon that will appear on the top bar.

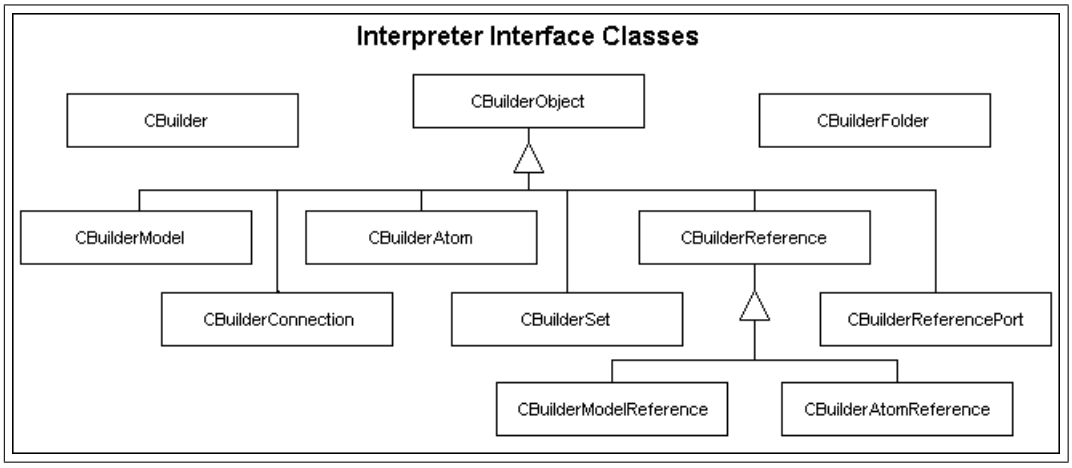


Figure 7: Classes in the Builder Object Network [14].

Implementing the interpreter was generally straightforward but there were some challenges. For instance, debugging was somewhat challenging because the code had to be run from within GME, so no sophisticated debugging tools were available. However, that issue was solved by showing debugging statements using programmatically-created dialogs. Another difficulty was in the implementation of the BPM gateways in Java, as they may lead to several different process paths depending on the execution of the decision logic. This was solved by implementing gateways to initially hold a list of all possible paths, then providing setters that get invoked during the execution of the decision logic and determine which path should actually be taken.

In terms of achievements, the current version of the interpreter supports the translation of all elements defined in the metamodel. The resulting Java code is also completely runnable, even though it lacks the actual decision logic. The interpreter immediately executes the generated Java code after translation, but this behaviour could be extended to enable the code to be serialized and saved for later loading and execution.

12 Critical Evaluation of GME

Throughout the project the team had a chance to analyse the advantages and disadvantages of GME from a more informed point of view. One one hand, GME is a very stable tool and throughout the project not a single bug was observed in it. The user interface is well organised and responsive, and the concept of dragging elements from the left-hand panel into the modelling area to create new instances made model assembly fast and intuitive. Learning how to use GME can be slightly complicated, but the tool is well documented in tutorials and also in the user manual.

On the other hand, GME needs a better integration between the metamodel and the model. For example, they cannot be simultaneously open in the same instance of GME, which makes it cumbersome to update inconsistencies or details in the metamodel that are noticed while creating a model. Another problem experienced by the team was that the automatic model update that happens as a result of changes to the metamodel often did not work, which meant that the model had to be manually updated via an XML importing operation. This turned out to be a nuisance even when performing relatively small metamodel changes.

There were several websites for the different versions of GME, which meant that it was quite easy to accidentally refer to outdated documentation. It would be nice if those respon-

sible for the GME project could group all of the documentation and downloads under the same website, organised by version. Finally, the customisation of icons did not work for the team, even after carefully following the tutorial. The mechanism for incorporating icons into the metamodel should be improved, since the team had to resort to manually setting the icons for each entity type in the model. Overall, the team considered GME to be a reliable and flexible modelling tool which offers good features and is intuitive to use, despite some minor inconveniences.

13 Critical Evaluation of Project Results

In this section the team insights regarding the work performed in this project are presented, organised by deliverable:

- **Metamodel:** the metamodel produced is a good starting point for a customised BPM language, but further analysis of the business model of a postal company is needed to develop a comprehensive solution. This analysis, however, would depend on having access to the inner workings of a large postal company such as NZ post. Another aspect of the metamodel which needs more work is the implementation of constraints. Even though basic constraints were added to the metamodel, more are required to guarantee that the order and combination of BPM elements always makes sense.
- **Model:** the model produced was expressive, but also a simplified version of what a real-life delivery request would entail. For instance, the current model does not recover from errors in validation or when choosing a transportation mode, instead ending the process with an error. In a real process, there should be recovery routines to provide the user with an opportunity to correct any problems. Another limitation of the model is that it exists in isolation, when in reality each process is likely to communicate with others by using the messaging capabilities offered by a BPM.
- **Java code:** the Java code created to represent the BPM is a flexible and extensible framework, but it is currently void of real functionality. This structure would have to be extended to support the needs of a real business process. Additionally, since this framework is meant to represent a service that is offered online, its design should incorporate some elements of web technologies. It must be noted that any major redesign of the Java BPM implementation would incur a redesign of the interpreter as well.

14 Work Distribution in Team

The work distribution of this project was even, with both team members participating in most tasks. The following table documents which team member has participated in which project activity:

Activity	Responsible Team Member
Decision of tool, language and concept to be modelled.	Ashleigh and Alex
Design of metamodel.	Ashleigh and Alex
Creation of metamodel.	Ashleigh
Design and implementation of OCL constraints in metamodel.	Ashleigh
Creation of model.	Alex
Design and creation of Java code.	Alex
Implementation of interpreter.	Alex
Writing and editing of report.	Ashleigh and Alex

15 Conclusion

This project clearly shows that model-driven development is useful because it enables non-technical users to directly record their knowledge of business requirements details using a standardised and non-technical language. Since each entity in the model corresponds to an already-implemented customised functionality module, the use of a BPM reduces the amount of code that has to be rewritten each time a process is created and updated.

By its very definition, the efficacy of the model-driven development approach is dependent on the flexibility of the modelling tool used. While GME is a good tool, it is geared towards users who understand the abstract concepts and intricacies of the field of modelling. This means that there exist opportunities for creating a tool that is geared towards non-technical users, but which also integrates with the work of technical users. Such a tool would be commercially viable, as it would improve stakeholder communication and reduce the amount of work needed during development.

The deliverables produced for this project are part of a proof-of-concept, but the framework built in Java could easily be populated with complex logic. Additionally, the Java data structure produced by the translation process is currently immediately executed, but this behaviour could be modified by adding the ability to save it for later execution. This project illustrates the relevance of the model-driven development paradigm, and evidences the productivity gains to be had from applying this technique to complex systems.

References

[1] Business Process Model and Notation. `http://en.wikipedia.org/wiki/Business_Process_Model_and_Notation`. Accessed on 28 September 2013.

[2] GME Documentation and Downloads Page. `https://forge.isis.vanderbilt.edu/gme/`. Accessed on 28 September 2013.

[3] Products and Services Offered by New Zealand Post. `http://www.nzpost.co.nz/products-services`. Accessed on 28 September 2013.

[4] AL-RAWAS, A., AND EASTERBROOK, S. M. *Communication problems in requirements engineering: a field study*. National Aeronautics and Space Administration, 1996.

[5] BÉZIVIN, J., BRUNETTE, C., CHEVREL, R., JOUAULT, F., AND KURTEV, I. Bridging the generic modeling environment (gme) and the eclipse modeling framework (emf). In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA* (2005), vol. 5, Citeseer.

[6] HAILPERN, B., AND TARR, P. Model-driven development: The good, the bad, and the ugly. *IBM systems journal 45*, 3 (2006), 451–461.

[7] HOOK, G. Business process modeling and simulation. In *Simulation Conference (WSC), Proceedings of the 2011 Winter* (2011), IEEE, pp. 773–778.

[8] INDULSKA, M., GREEN, P., RECKER, J., AND ROSEMANN, M. Business process modeling: Perceived benefits. In *Conceptual Modeling-ER 2009*. Springer, 2009, pp. 458–471.

[9] KOEHLER, J., TIRENNI, G., AND KUMARAN, S. From business process model to consistent implementation: A case for formal verification methods. In *Enterprise Distributed Object Computing Conference, 2002. EDOC'02. Proceedings. Sixth International* (2002), IEEE, pp. 96–106.

[10] LEDECZI, A., MAROTI, M., BAKAY, A., KARSAI, G., GARRETT, J., THOMASON, C., NORDSTROM, G., SPRINKLE, J., AND VOLGYESI, P. The generic modeling environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary* (2001), vol. 17.

[11] MELLOR, S. J., CLARK, A. N., AND FUTAGAMI, T. Model-driven development. *IEEE software* (2003), 14–18.

[12] SELIC, B. The pragmatics of model-driven development. *Software, IEEE 20*, 5 (2003), 19–25.

[13] VAN DER AALST, W. M., DUMAS, M., GOTTSCHALK, F., TER HOFSTEDE, A. H., LA ROSA, M., AND MENDLING, J. Preserving correctness during business process model configuration. *Formal Aspects of Computing 22*, 3-4 (2010), 459–482.

[14] WHITE, S. A. Introduction to bpmn. *IBM Cooperation* (2004), 2008–029.